



Instructivo de GLPK

Modelos y Optimización I

Redactado por Tomás Bruno

Facultad de Ingeniería de la Universidad de Buenos Aires

Versión 1.0 - Abril 2012

Índice

1. Introducción	2
1.1. Instalación	2
1.1.1. Ubuntu/Linux	2
1.1.2. Windows	2
2. Uso de GLPK	3
2.1. Problema	3
2.2. Modelo	4
2.3. Resolución básica utilizando GLPK	5
2.4. Ejecución y análisis de una corrida	6
2.5. Salida de la consola	9
3. Separación de los datos del modelo	10
4. Separación de datos del modelo en GLPK	13
4.1. Modelo	13
4.2. Sección de datos	15
4.3. Ejecución	16
5. Funciones	17
6. Limitando el tiempo de ejecución	18

1. Introducción

El presente documento pretende ser introductorio al uso del programa *GLPK (GNU Linear Programming Kit)* que es utilizado para la resolución de problemas de programación lineal.

1.1. Instalación

1.1.1. Ubuntu/Linux

Para instalar GLPK, se debe ingresar en una consola el siguiente comando:

```
sudo apt-get install glpk-utils
```

Luego de instalado, podremos utilizar el comando de consola “*glpsol*” . Para verificar que la instalación se realizó correctamente pueden ingresar el comando

```
glpsol -v
```

y observar la versión instalada.

En la ubicación donde se instala, se podrá encontrar la documentación del programa referenciada como [1] y [2].

1.1.2. Windows

Desde sistemas operativos Windows se puede utilizar *GUSEK (GLPK Under Scite Extended Kit)*. *GUSEK* es un programa que provee una interfaz gráfica mediante la integración de *GLPK* a un editor de texto llamado *Scite*.

El programa se debe descargar de <http://gusek.sourceforge.net/>. Tal como dice el instructivo, *GUSEK* es una aplicación portable. Una vez que se baja y se extrae el contenido de la carpeta, solo se debe correr el archivo *gusek.exe* para ejecutarlo. En la misma carpeta, se podrá encontrar la documentación del programa referenciada en este instructivo como [1] y [2].

2. Uso de GLPK

Para ilustrar el uso de GLPK, se utilizará como ejemplo el ejercicio tipo 5.2 cuyo enunciado se incluye a continuación.

2.1. Problema

En una fábrica de medias se desea analizar la operación de un sector integrado por tres equipos $E1$, $E2$, $E3$ donde se procesan los productos A , B , C . Los tiempos de proceso de los productos son los del siguiente cuadro, medidos en horas de equipo por docena de producto.

	A	B	C
Equipo 1	0,8	0,8	0,3
Equipo 2	0,6	1,2	0
Equipo 3	0,6	1,0	0,6

Se ha determinado además, la disponibilidad mensual de cada uno de los equipos. Esta importa respectivamente 160, 180 y 110 horas. Asimismo, se estima en 100 docenas mensuales la cantidad demandada máxima del producto A , y en 120 docenas mensuales la cantidad demandada máxima del producto B .

Por otra parte, la Dirección de la empresa desea producir como mínimo 80 docenas mensuales del producto B . El margen de beneficio de cada producto es de 50\$ por docena de A , 40\$ por docena de B y 30\$/docena de C . El programa óptimo es el que hace máximo el margen total de beneficio.

2.2. Modelo

Procesamiento

$$0,8A + 0,8B + 0,3C < 160 \quad (\text{procEq1})$$

$$0,6A + 1,2B < 180 \quad (\text{procEq2})$$

$$0,6A + 1B + 0,6C < 110 \quad (\text{procEq3})$$

Demandas máximas y mínimas

$$A < 100 \quad (\text{demMaxA})$$

$$B < 120 \quad (\text{demMaxB})$$

$$B > 80 \quad (\text{demMinB})$$

Definición del funcional

$$MAX \rightarrow Z = 50\$A + 40\$B + 30\$C$$

Figura 1: Modelo

2.3. Resolución básica utilizando GLPK

La figura 2 muestra la forma más directa de expresar el modelo. Se puede observar que cada una de las ecuaciones del modelo 1 está explícitamente expresada.

```
# Resolución 1 - Contenido del archivo 1.mod.
/* Declaración de variables */
var A >= 0;
var B >= 0;
var C >= 0;

/* Definición del funcional */
maximize z: 50 * A + 40 * B + 30 * C ;

/* Restricciones */
/* Procesamiento de cada equipo */
s.t. procEq1: 0.8 * A + 0.8 * B + 0.3 * C <= 160;
s.t. procEq2: 0.6 * A + 1.2 * B <= 180;
s.t. procEq3: 0.6 * A + 1 * B + 0.6 * C <= 110;

/* Demandas máximas y mínimas */

s.t. demMaxA: A <= 100;
s.t. demMaxB: B <= 120;
s.t. demMinB: B >= 80;

end;
```

Figura 2: contenido del archivo “1.mod”

Lo primero que se puede observar es la sección de variables. En esta sección se indica la existencia de 3 variables A, B y C que representarán la cantidad a producir (y vender) de cada uno de los productos. Pese a tratarse de un problema de programación lineal y que en consecuencia todas las variables deben ser iguales o mayor que 0, es necesario indicar que cada variable es “ ≥ 0 ”.

En segundo lugar, está definido el funcional y por último se encuentra la sección de las restricciones. Cada restricción comienza con “s.t.” e incluye un rótulo con el cual se podrá identificar la restricción en la solución.

2.4. Ejecución y análisis de una corrida

Para realizar la corrida del modelo, desde una consola se ingresa el siguiente comando:

```
glpsol -m 1.mod -o 1.sol
```

Figura 3: Corrida en consola

En caso de estar utilizando Gusek, una vez abierto el archivo del modelo, se pueden configurar los parámetros como lo muestra la Figura 4. Para ejecutarlo se debe ejecutar la opción “go”.

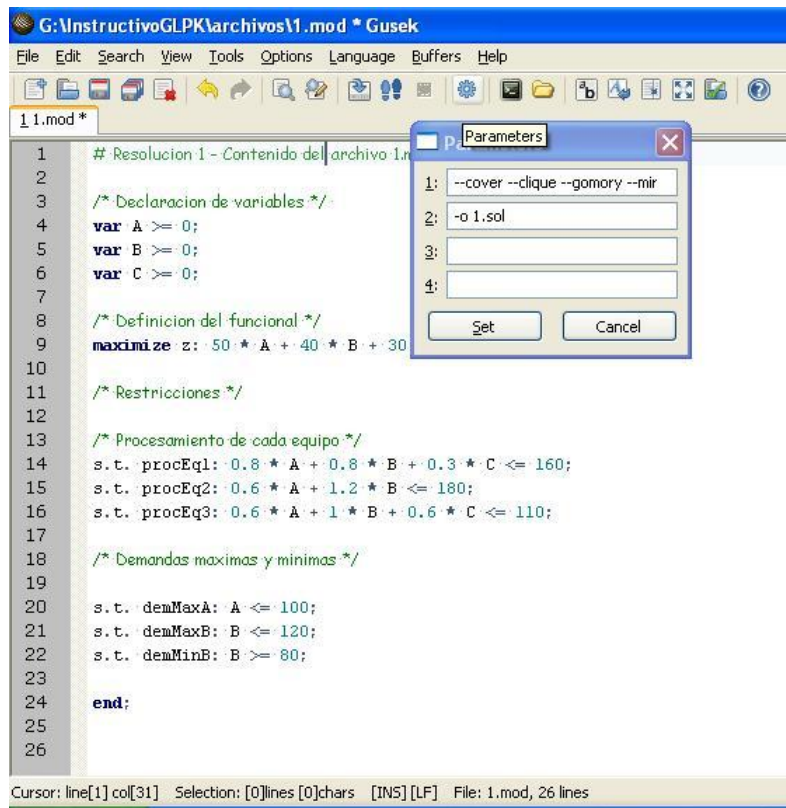


Figura 4: Corrida en Gusek

La opción “m” sirve para indicar cuál es el modelo que se utilizará mientras que la opción “o” indica el nombre del archivo donde se imprimirá la salida. En la terminal (o en la solapa derecha de Gusek) se imprime información cuya utilidad se analizará más adelante. Por el momento, alcanza con observar el contenido del archivo *1-sol*:

```

1 Problem: 1
2 Rows: 7
3 Columns: 3
4 Non-zeros: 14
5 Status: OPTIMAL
6 Objective: z = 5700 (MAXimum)
7
8 No. Row name St Activity Lower bound Upper bound Marginal
9 -----
10 1 z B 5700
11 2 procEq1 B 104 160
12 3 procEq2 B 126 180
13 4 procEq3 NU 110 110 83.3333
14 5 demMaxA B 50 100
15 6 demMaxB B 80 120
16 7 demMinB NL 80 80 -43.3333
17
18 No. Column name St Activity Lower bound Upper bound Marginal
19 -----
20 1 A B 50 0
21 2 B B 80 0
22 3 C NL 0 0 -20
23
24 Karush-Kuhn-Tucker optimality conditions:
25
26 KKT.PE: max.abs.err = 0.00e+00 on row 0
27 max.rel.err = 0.00e+00 on row 0
28 High quality
29
30 KKT.PB: max.abs.err = 0.00e+00 on row 0
31 max.rel.err = 0.00e+00 on row 0
32 High quality
33
34 KKT.DE: max.abs.err = 7.11e-15 on column 1
35 max.rel.err = 7.04e-17 on column 1
36 High quality
37
38 KKT.DB: max.abs.err = 0.00e+00 on row 0
39 max.rel.err = 0.00e+00 on row 0
40 High quality
41
42 End of output

```

Figura 5: Resultado de la corrida

En la corrida podemos identificar los siguientes elementos.

- *Problem*: Nombre del archivo que contiene el modelo.
- *Rows*: Cantidad de filas de la matriz. (Cantidad de restricciones más el funcional)
- *Columns*: Cantidad de columnas. Existe una columna por cada variable
- *Non-zeros*: Cantidad de elementos en la matriz distintos de 0. (Recordar que el funcional es una fila más en la matriz).
- *Status*: Tipo de solución. En este caso existe una única solución.
- *Objective*: Valor del funcional

Luego de la descripción general de la solución, se presentan dos tablas en donde se puede observar el estado de cada restricción y cada variable. La primer tabla provee información acerca de las restricciones. Para cada una podemos ver su estado, su nivel de actividad y su valor marginal.

Por ejemplo, podemos ver que la restricción relativa al procesamiento del equipo 1, *procEq1*, tiene un nivel de actividad de 104. Es decir, se están consumiendo 104hs del equipo 1. La disponibilidad total de tiempo de este equipo es de 160hs. En consecuencia, esta restricción se encuentra inactiva. Al sobrnarnos horas de equipo 1, el valor marginal asociado es 0.

Distinto es el caso de la restricción correspondiente al procesamiento del equipo 3, *procEq3*, donde se puede ver que el nivel de actividad es igual a su cota **superior**. En este caso existe un valor marginal de 83,3\$. Es decir, si aumentara la cota superior en una unidad, que en este caso sería tener una hora más de procesamiento de equipo 3 disponible, ganaríamos 83,3\$ más.

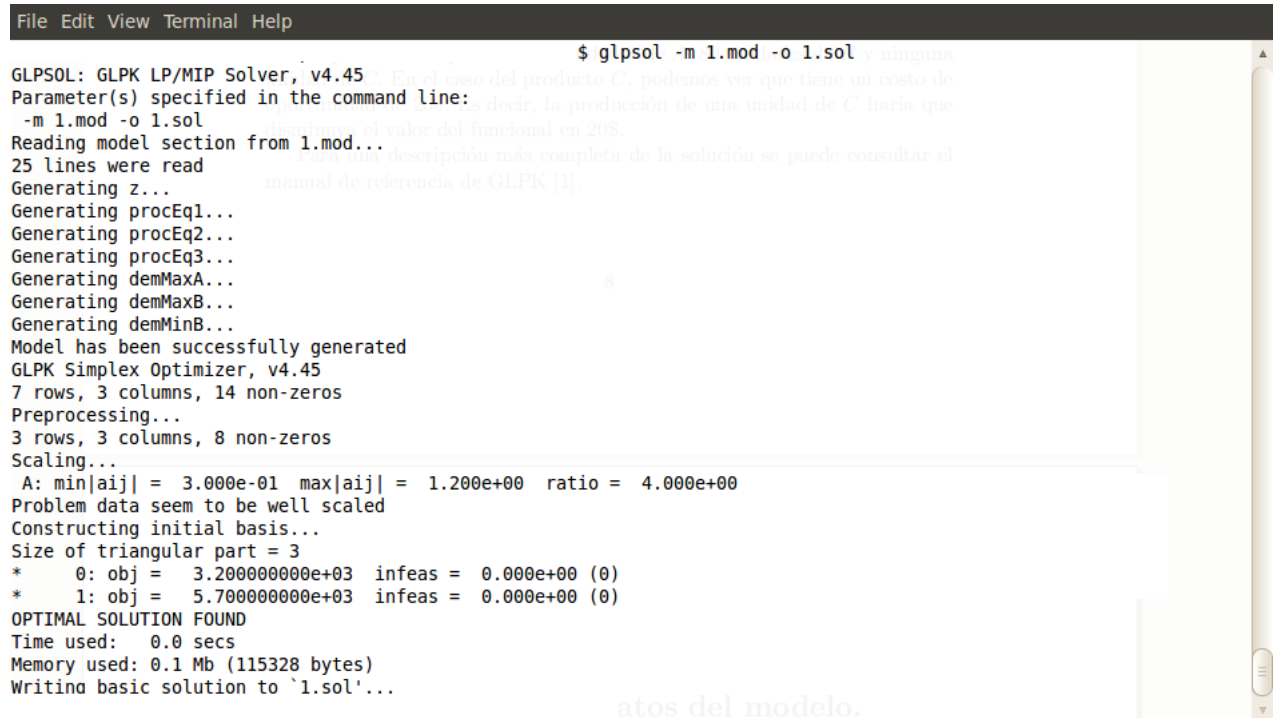
El tercer caso que podemos identificar es el de la restricción de demanda mínima del producto *B*, *demMinB*. El nivel de actividad de esta restricción es igual a su cota **inferior**. Si aumentara en una unidad la restricción, estaría obligado a producir una unidad más y en consecuencia el funcional disminuiría en 43,3\$

En la segunda tabla se puede identificar el valor de cada variable. El modelo indica que conviene producir 50 unidades de *A*, 80 unidades de *B* y ninguna unidad de *C*. En el caso del producto *C*, podemos ver que tiene un costo de oportunidad de 20\$. Es decir, la producción de una unidad de *C* haría que disminuya el valor del funcional en 20\$.

Para una descripción más completa de la solución se puede consultar el manual de referencia de GLPK [1].

2.5. Salida de la consola

La salida de la consola de la primer corrida (3) es la que se puede observar a continuación.



```
File Edit View Terminal Help
$ glpsol -m 1.mod -o 1.sol
GLPSOL: GLPK LP/MIP Solver, v4.45
Parameter(s) specified in the command line:
-m 1.mod -o 1.sol
Reading model section from 1.mod...
25 lines were read
Generating z...
Generating procEq1...
Generating procEq2...
Generating procEq3...
Generating demMaxA...
Generating demMaxB...
Generating demMinB...
Model has been successfully generated
GLPK Simplex Optimizer, v4.45
7 rows, 3 columns, 14 non-zeros
Preprocessing...
3 rows, 3 columns, 8 non-zeros
Scaling...
A: min|aij| = 3.000e-01 max|aij| = 1.200e+00 ratio = 4.000e+00
Problem data seem to be well scaled
Constructing initial basis...
Size of triangular part = 3
* 0: obj = 3.200000000e+03 infeas = 0.000e+00 (0)
* 1: obj = 5.700000000e+03 infeas = 0.000e+00 (0)
OPTIMAL SOLUTION FOUND
Time used: 0.0 secs
Memory used: 0.1 Mb (115328 bytes)
Writing basic solution to `1.sol'...
```

Figura 6: Salida de la terminal

Esta salida contiene información acerca de cómo se llegó a la solución que finalmente se muestra en la Figura 5. En la sección 6 se analizará la importancia de esta información.

3. Separación de los datos del modelo

¿Qué pasaría si el día de mañana se decidiera la inclusión de un nuevo producto D ? Este producto al igual que los otros, sería procesado por los equipos, con lo cual para cada máquina tendría un determinado tiempo de procesamiento. A su vez, también podría tener una demanda máxima y mínima. Siguiendo nuestra forma de resolverlo en la Figura 1, en cada una de las restricciones de procesamiento tendríamos que agregar el producto D multiplicado por el tiempo que insume en cada máquina. También agregaríamos las ecuaciones de demanda máxima y mínima. Hacer estos pocos cambios en un modelo muy pequeño como el que estamos analizando es sencillo, ¿Pero qué pasaría si en vez de tener 3 equipos tuviéramos 100?

A grandes rasgos, el problema sobre el cual estamos trabajando tiene 2 tipos de restricciones: el procesamiento de los productos que realiza cada producto y la existencia de demandas máximas y mínimas. Si lo resolviéramos genéricamente, quedaría un modelo como el siguiente:

Procesamiento

$$\sum_{PRODUCTOS} \underbrace{TIEMPO_{prod, equipo}}_{Constante} \cdot prod < \underbrace{DISP_{equipo}}_{Constante} \quad (\forall equipo)$$

Demandas máximas y mínimas

$$prod < \underbrace{DEM_MAX_{prod}}_{Constante} \quad (\forall prod)$$

$$prod > \underbrace{DEM_MIN_{prod}}_{Constante} \quad (\forall prod)$$

Definición del funcional

$$MAX \rightarrow Z = \sum_{PRODUCTOS} \underbrace{PRECIO_{prod}}_{Constante} \cdot prod$$

Figura 7: Modelo genérico

Es importante notar que en el modelo ilustrado en la Figura 7, en ningún momento se hace referencia a productos, equipos, precios o tiempos de procesamiento específicos. Solamente se establece una estructura. Lo que todavía resta sería indicar para que datos queremos resolver el problema. La definición de los datos podría estar dada de la siguiente manera.

Equipo	A	B	C
1	0,8	0,8	0,3
2	0,6	1,2	0
3	0,6	1,0	0,6

Equipo	Disponibilidad
1	160
2	180
3	110

Prod	Dem Max	Dem Min
A	100	-
B	120	80
C	-	-

Producto	Precio
A	50\$
B	40\$
C	30\$

Figura 8: Sección de datos

Entonces si volviéramos a la pregunta original, ¿Qué pasaría si en vez de tener 3 equipos tuviéramos 100? Nos daríamos cuenta que ahora que tenemos los datos separados del modelo, lo único que tendríamos que hacer es generar las entradas en las tablas correspondientes. **En ningún momento estaríamos alterando las restricciones del modelo.**

Los mayores beneficios de realizar esta separación de los datos del modelo se pueden ver en otro tipo de problemas, entre ellos los combinatorios. En un problema del viajante por ejemplo, si aumentáramos de 100 a 101 ciudades, tendríamos que agregar 202 restricciones y modificar otras miles de ecuaciones. Si en cambio tuviéramos separado el modelo de los datos, lo único que tendríamos que agregar es una fila con las distancias a cada ciudad.

Todavía queda un problema con esta idea de tener al modelo separado de los datos: ¿Qué pasa por ejemplo con el producto C que no tiene una demanda máxima? La solución más sencilla en este caso sería definir un valor suficientemente grande que sepamos que nunca podrá valer. Sin embargo, este tipo de solución puede traer problemas. Existe una alternativa que consiste en definir una *constante* a la que le asignaremos un valor de uno o cero dependiendo de si la variable tiene o no demanda máxima. El nuevo modelo quedaría de la siguiente forma:

Procesamiento

$$\sum_{PRODUCTOS} \underbrace{TIEMPO_{prod, equipo} \cdot prod}_{Constante} < \underbrace{DISP_{equipo}}_{Constante} \quad (\forall equipo)$$

Demandas máximas y mínimas

$$\underbrace{TIENE_DEM_MAX_{prod} \cdot prod}_{Constante} < \underbrace{DEM_MAX_{prod}}_{Constante} \quad (\forall prod)$$

$$\underbrace{TIENE_DEM_MIN_{prod} \cdot prod}_{Constante} > \underbrace{DEM_MIN_{prod}}_{Constante} \quad (\forall prod)$$

Definición del funcional

$$MAX \rightarrow Z = \sum_{PRODUCTOS} \underbrace{VENTA_{prod} \cdot prod}_{Constante}$$

Figura 9: Modelo genérico refinado

Habiendo definido estas constantes, podemos plantear nuestra nueva sección de datos.

Equipo	A	B	C
1	0,8	0,8	0,3
2	0,6	1,2	0
3	0,6	1,0	0,6

Equipo	Disponibilidad
1	160
2	180
3	110

Prod	Tiene Dem Max	Dem Max
A	1	100
B	1	120
C	0	0

Prod	Tiene Dem Min	Dem Min
A	0	0
B	1	80
C	0	0

Producto	Precio
A	50\$
B	40\$
C	30\$

Figura 10: Sección de datos refinada

4. Separación de datos del modelo en GLPK

En la siguiente sección se ejemplificará como se hace para separar los datos del modelo en GLPK. No obstante, debe quedar en claro que la idea de realizar esta separación no es ni propia ni única de GLPK. Otros softwares como CPLEX también lo permiten.

4.1. Modelo

Para expresar el modelo genérico (9), se lo podría implementar como lo muestra la Figura 11.

Lo primero que se puede observar es la existencia de Conjuntos (Sets). Un conjunto estará formado por elementos. En el modelo se podrá iterar sobre estos elementos, pero no hace falta indicar específicamente que elementos son.

Luego de la definición de los conjuntos, viene la definición de parámetros. Cada parámetro representa un conjunto de constantes. Para cada parámetro definiremos una tabla en la sección de datos. Se puede observar que para cada parámetro, se indica la cantidad de constantes que se definen. Por ejemplo, si se encuentra definida la siguiente línea.

```
param DISP {j in EQUIPOS};
```

el modelo esperará que en la sección de datos este definida una disponibilidad para cada equipo.

La declaración de variables se hace para cada elemento del conjunto de productos.

En el funcional se puede ver la aparición de la función sumatoria. A esta función se le pasa como parámetro el o los conjuntos sobre los cuales debe iterar. Por último, también se puede observar que en la restricción de *utilización de los equipos* la forma en la que se expresa el \forall . Ingresando un conjunto antes del ':', el programa crea la restricción para cada uno de los elementos del conjunto indicado.

Si quisiéramos excluir a un elemento determinado (por ejemplo al producto C de la demanda máxima), podríamos escribir la línea de la siguiente forma:

```
s.t. max{i in PRODUCTOS: i<>'C'}: TIENE_DEM_MAX[i] * prods[i]
                                     <= DEM_MAX[i];
```

De todas formas es necesario notar que al incluir sentencias como esta última estaríamos generando cierto acoplamiento entre los datos y el modelo.

```

# Conjuntos
set PRODUCTOS;
set EQUIPOS;
# Parámetros
param DISP{j in EQUIPOS};
param UTILIZA{j in EQUIPOS, i in PRODUCTOS};
param VENTA{i in PRODUCTOS};
param DEM_MAX{i in PRODUCTOS};
param TIENE_DEM_MAX{i in PRODUCTOS};
param DEM_MIN{i in PRODUCTOS};
param TIENE_DEM_MIN{i in PRODUCTOS};

#Definición de variables
var prods{i in PRODUCTOS} >= 0;

/* Funcional */
maximize z: sum{i in PRODUCTOS} VENTA[i] * prods[i];

/* Restricciones */

# Utilización de los equipos
s.t. proc{j in EQUIPOS}: sum{i in PRODUCTOS} UTILIZA[j,i] * prods[i]
<= DISP[j];

# Demandas.
s.t. max{i in PRODUCTOS}: TIENE_DEM_MAX[i] * prods[i] <= DEM_MAX[i];
s.t. min{i in PRODUCTOS}: TIENE_DEM_MIN[i] * prods[i] >= DEM_MIN[i];

end;

```

Figura 11: Contenido del archivo “2.mod”

4.2. Sección de datos

```
# Data Section
data;

set PRODUCTOS := A B C ;
set EQUIPOS := eq1 eq2 eq3 ;

param UTILIZA: A B C:=
eq1 0.8 0.8 0.3
eq2 0.6 1.2 0.0
eq3 0.6 1.0 0.6;
param DISP :=
eq1 160
eq2 180
eq3 110;
param VENTA :=
A 50
B 40
C 30;
param TIENE_DEM_MAX :=
A 1
B 1
C 0;
param DEM_MAX :=
A 100
B 120
C 0;
param TIENE_DEM_MIN :=
A 0
B 1
C 0;
param DEM_MIN :=
A 0
B 80
C 0;
end;
```

Figura 12: Contenido del archivo “2.dat”

4.3. Ejecución

Para realizar la corrida, desde la una consola se ingresa el siguiente comando:

```
glpsol -m 2.mod -d 2.dat -o 2.sol
```

Podemos observar que el resultado es el mismo que en la figura 5.

```
1 Problem:      2
2 Rows:        10
3 Columns:     3
4 Non-zeros:   14
5 Status:      OPTIMAL
6 Objective:   z = 5700 (MAXimum)
7
8 No.  Row name  St  Activity  Lower bound  Upper bound  Marginal
9 -----
10 1 z          B    5700
11 2 proc[eq1]  B    104      160
12 3 proc[eq2]  B    126      180
13 4 proc[eq3]  NU   110      110      83.3333
14 5 max[A]     B    50       100
15 6 max[B]     B    80       120
16 7 max[C]     B    0        -0
17 8 min[A]     B    0        -0
18 9 min[B]     NL   80       80      -43.3333
19 10 min[C]    B    0        -0
20
21 No.  Column name  St  Activity  Lower bound  Upper bound  Marginal
22 -----
23 1 prods[A]    B    50       0
24 2 prods[B]    B    80       0
25 3 prods[C]    NL   0        0      -20
```

Figura 13: Corrida del modelo 2

5. Funciones

El lenguaje en el que fueron expresados todos los ejemplos de este instructivo es GMPL. GMPL (también conocido como MathProg) es un lenguaje creado para expresar modelos matemáticos. Tiene determinadas funciones que pueden ayudar en la tarea de modelación. Algunas de ellas son:

- **card(X)**: Devuelve la cardinalidad(cantidad de elementos) del CONJUNTO(SET) que recibe como parámetro.

Ejemplo de uso: en el ejercicio 5.2, definiendo los parámetros como lo muestra la Figura 12,

```
card(PRODUCTOS)
```

la función devolvería 3 ya que esta es la cantidad de elementos del conjunto *PRODUCTOS*.

- **gmtime()**: Devuelve la cantidad de segundos desde 00:00:00 01/01/1970

.

- **Uniform(a, b)**: Devuelve un número al azar entre *a* y *b* (distribución uniforme).

- **max(x_1, x_2, \dots, x_n)**: Devuelve el mayor de los valores que recibe como parámetro.

Ejemplo de uso: en el ejercicio 5.2, definiendo los parámetros como lo muestra la Figura 12,

```
max(VENTA['A'], VENTA['B'], VENTA['C'])
```

la función devolvería 50 ya que este es el mayor precio de venta. Para escribirlo genéricamente podríamos expresarlo a través del operador `max`.

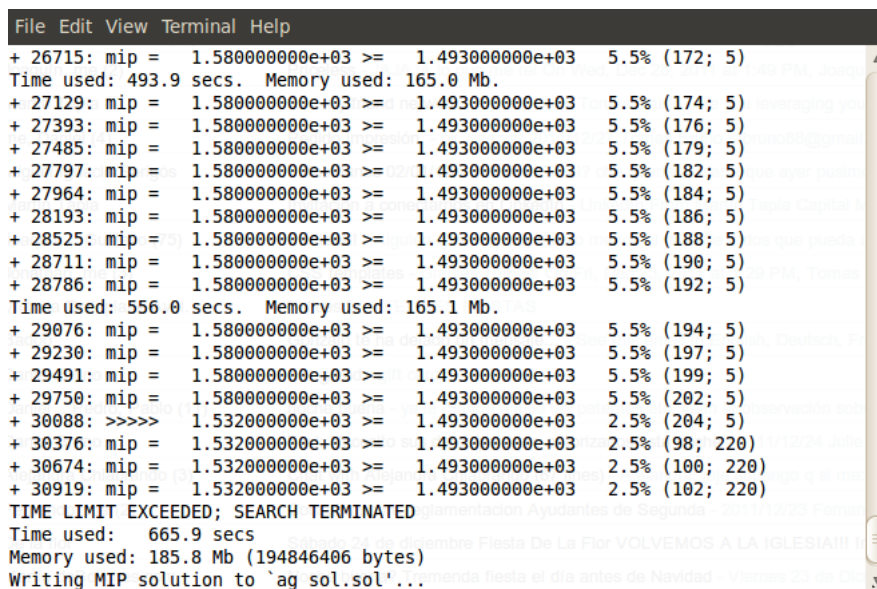
```
max{i in PRODUCTOS} VENTA[i]
```

El listado completo de funciones se puede leer en la documentación de GMPL [2].

6. Limitando el tiempo de ejecución

Es posible que al intentar resolver problemas combinatorios de gran escala, el tiempo de ejecución ascienda a días (o peor aún). En consecuencia, GLPK nos provee una opción para limitar el tiempo de búsqueda de la solución. Para hacerlo, al ejecutar la corrida debemos ingresar la opción `--tmlim nnn` donde `nnn` representa la cantidad de segundos que se desea establecer como límite. Una vez superado ese límite el programa deja de explorar el recinto de soluciones.

A continuación se muestra el final de una salida en la cual se establecieron 600 segundos como límite de tiempo.



```
File Edit View Terminal Help
+ 26715: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (172; 5)
Time used: 493.9 secs. Memory used: 165.0 Mb.
+ 27129: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (174; 5)
+ 27393: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (176; 5)
+ 27485: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (179; 5)
+ 27797: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (182; 5)
+ 27964: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (184; 5)
+ 28193: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (186; 5)
+ 28525: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (188; 5)
+ 28711: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (190; 5)
+ 28786: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (192; 5)
Time used: 556.0 secs. Memory used: 165.1 Mb.
+ 29076: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (194; 5)
+ 29230: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (197; 5)
+ 29491: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (199; 5)
+ 29750: mip = 1.580000000e+03 >= 1.493000000e+03 5.5% (202; 5)
+ 30088: >>>> 1.532000000e+03 >= 1.493000000e+03 2.5% (204; 5)
+ 30376: mip = 1.532000000e+03 >= 1.493000000e+03 2.5% (98; 220)
+ 30674: mip = 1.532000000e+03 >= 1.493000000e+03 2.5% (100; 220)
+ 30919: mip = 1.532000000e+03 >= 1.493000000e+03 2.5% (102; 220)
TIME LIMIT EXCEEDED; SEARCH TERMINATED
Time used: 665.9 secs
Memory used: 185.8 Mb (194846406 bytes)
Writing MIP solution to `ag_sol.sol'...
```

Figura 14: Salida de la consola con límite de tiempo

Es interesante observar la última línea cuando el programa seguía buscando soluciones.

```
+ 30919: mip = 1.532000000e+03 >= 1.493000000e+03 2.5% (102; 220)
```

- 30919 indica la cantidad de iteraciones realizadas por el momento.
- 1532 es el valor actual del funcional.
- 1493 es la mejor solución que podría llegar a encontrarse (problema de minimización).

- 2,5% es la diferencia porcentual entre el valor obtenido y la mejor solución que podría llegar a encontrarse.

Referencias

- [1] MAKHORIN, Andrew. GNU Linear Programming Kit. *Reference Manual*. Version 4.45. 2010.
- [2] MAKHORIN, Andrew. Modeling Language GNU MathProg. *Language Reference*. Version 4.45. 2010.
- [3] Página principal.
<http://www.gnu.org/software/glpk/>
- [4] WikiBook *GLPK*
<http://en.wikibooks.org/wiki/GLPK>
- [5] CERON, Rodrigo. The GNU Linear Programming Kit. 2006.
<http://www.ibm.com/developerworks/linux/library/l-glpk1/index.html>